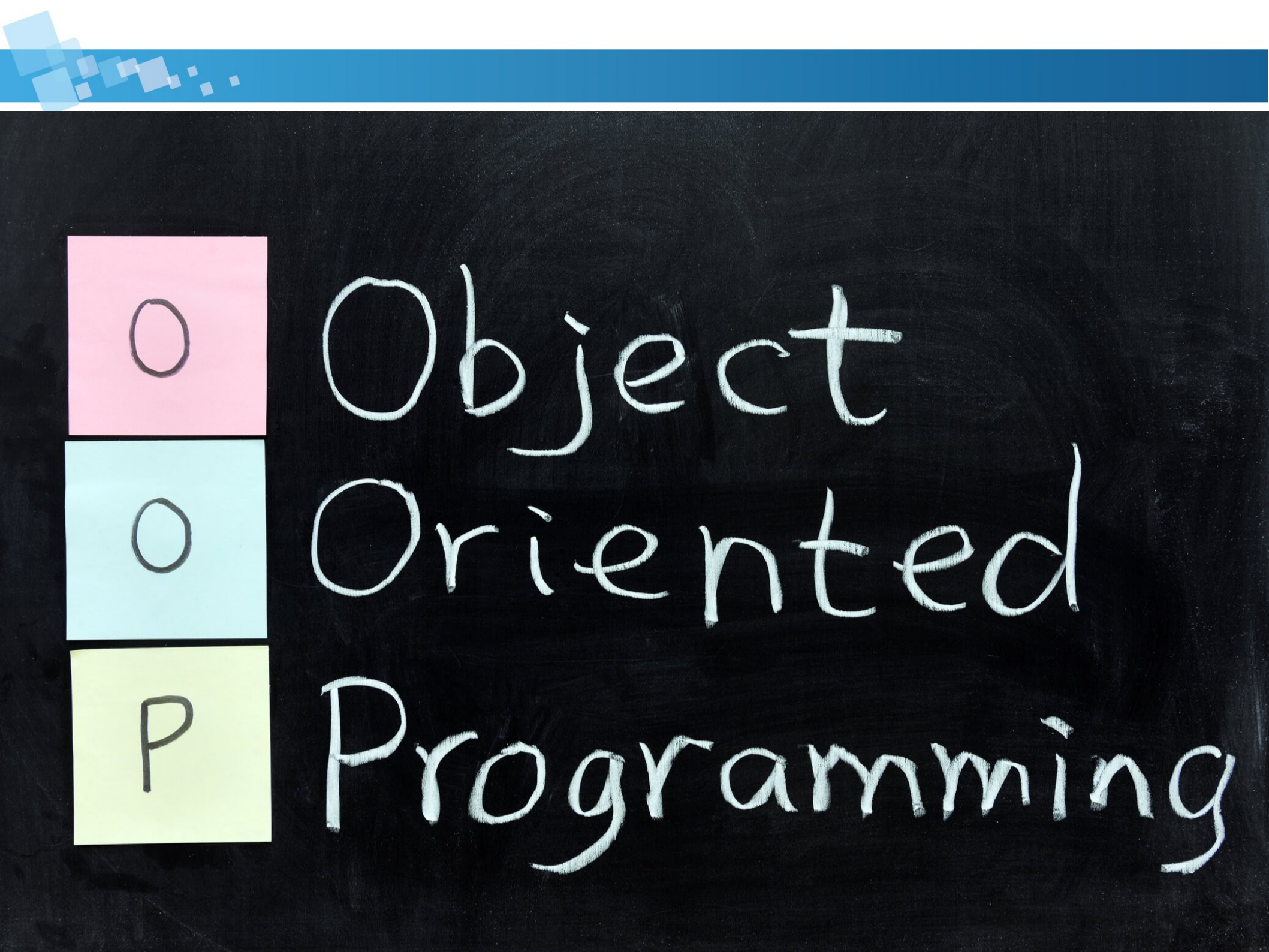
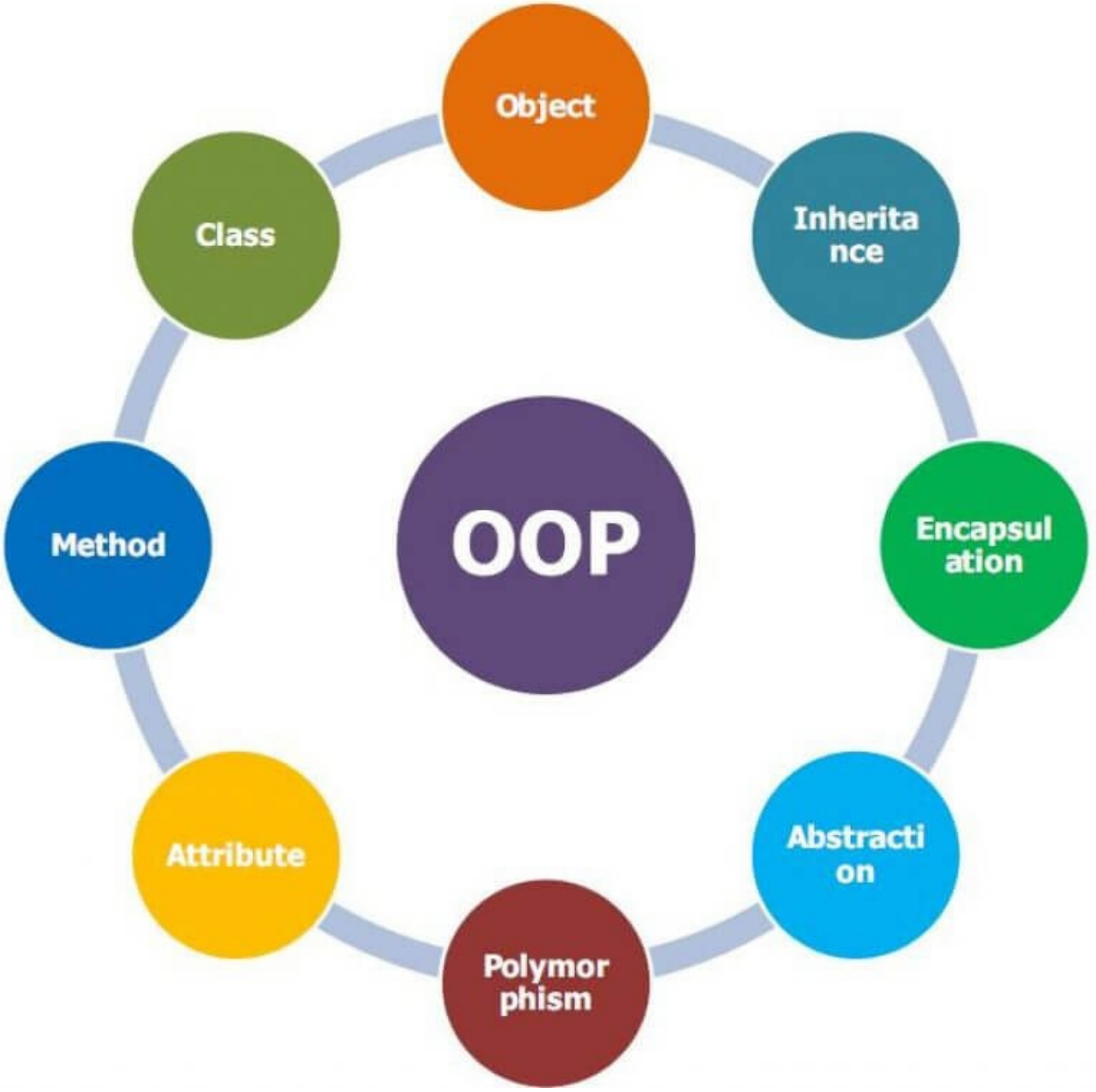
ООП в Python

Часть I


O
O
P
Object
Oriented
Programming

Определение ООП

Объе́ктно-ориентированное программирование (ООП) — методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определённого класса, а классы образуют иерархию наследования.


Object
Inherita nce
Encapsul ation
Abstracti on
Polymor phism
Attribute
Method
Class
OOP

Основные принцыпы ООП

- Инкапсуляция - сокрытие реализации.
- Наследование - создание новой сущности на базе уже существующей.
- Полиморфизм - возможность иметь разные формы для одной и той же сущности.
- Абстракция - набор общих характеристик.
- Посылка сообщений - форма связи, взаимодействия между сущностями.
- Переиспользование - повторное использование кода.

Инкапсуляция

- Инкапсуляция – это свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе и скрыть детали реализации от пользователя, открыв только то, что необходимо при последующем использовании.

- Цель инкапсуляции — уйти от зависимости внешнего интерфейса класса (то, что могут использовать другие классы) от реализации. Чтобы малейшее изменение в классе не влекло за собой изменение внешнего поведения класса.

Наследование

Наследование – это свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью.

Класс, от которого производится наследование, называется предком, базовым или родительским. Новый класс – потомком, наследником или производным классом.

Полиморфизм

Полиморфизм – это свойство системы использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

Преимуществом полиморфизма является то, что он помогает снижать сложность программ, разрешая использование одного и того же интерфейса для задания единого набора действий. Выбор же конкретного действия, в зависимости от ситуации, возлагается на компилятор языка программирования. Отсюда следует ключевая особенность полиморфизма - использование объекта производного класса, вместо объекта базового (потомки могут изменять родительское поведение, даже если обращение к ним будет производиться по ссылке родительского типа).

Абстракция данных

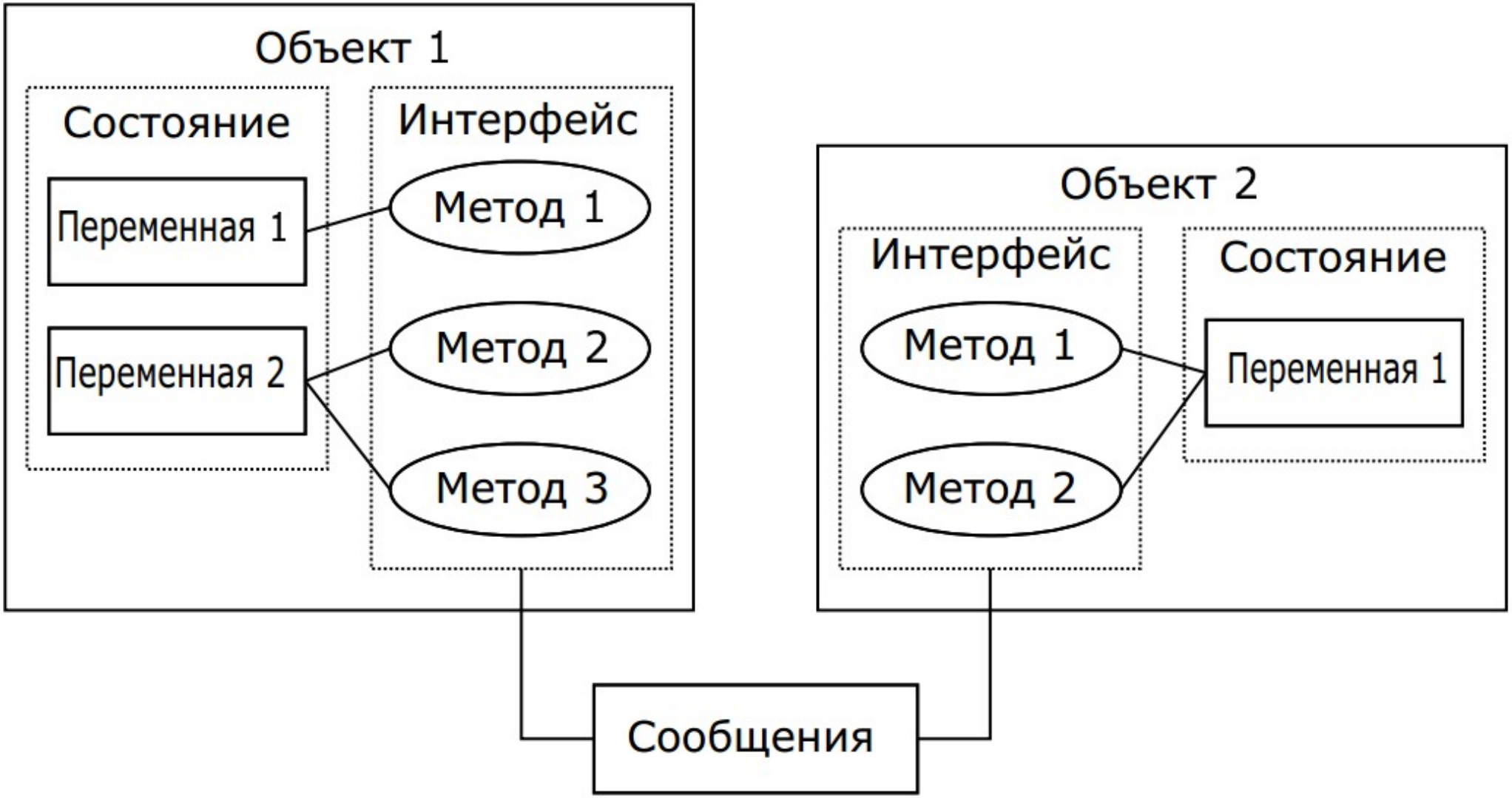
Абстрагирование означает выделение значимых характеристик и исключение из рассмотрения частных и незначимых. В ООП рассматривают лишь абстракцию данных (нередко называя её просто «абстракцией»), подразумевая набор наиболее общих характеристик объекта, доступных остальной программе.

Пример:

Представьте, что водитель едет в автомобиле по оживлённому участку движения. Понятно, что в этот момент он не будет задумываться о химическом составе краски автомобиля, особенностях взаимодействия шестерён в коробке передач или влияния формы кузова на скорость . Однако, руль, педали, указатель поворота он будет использовать регулярно.

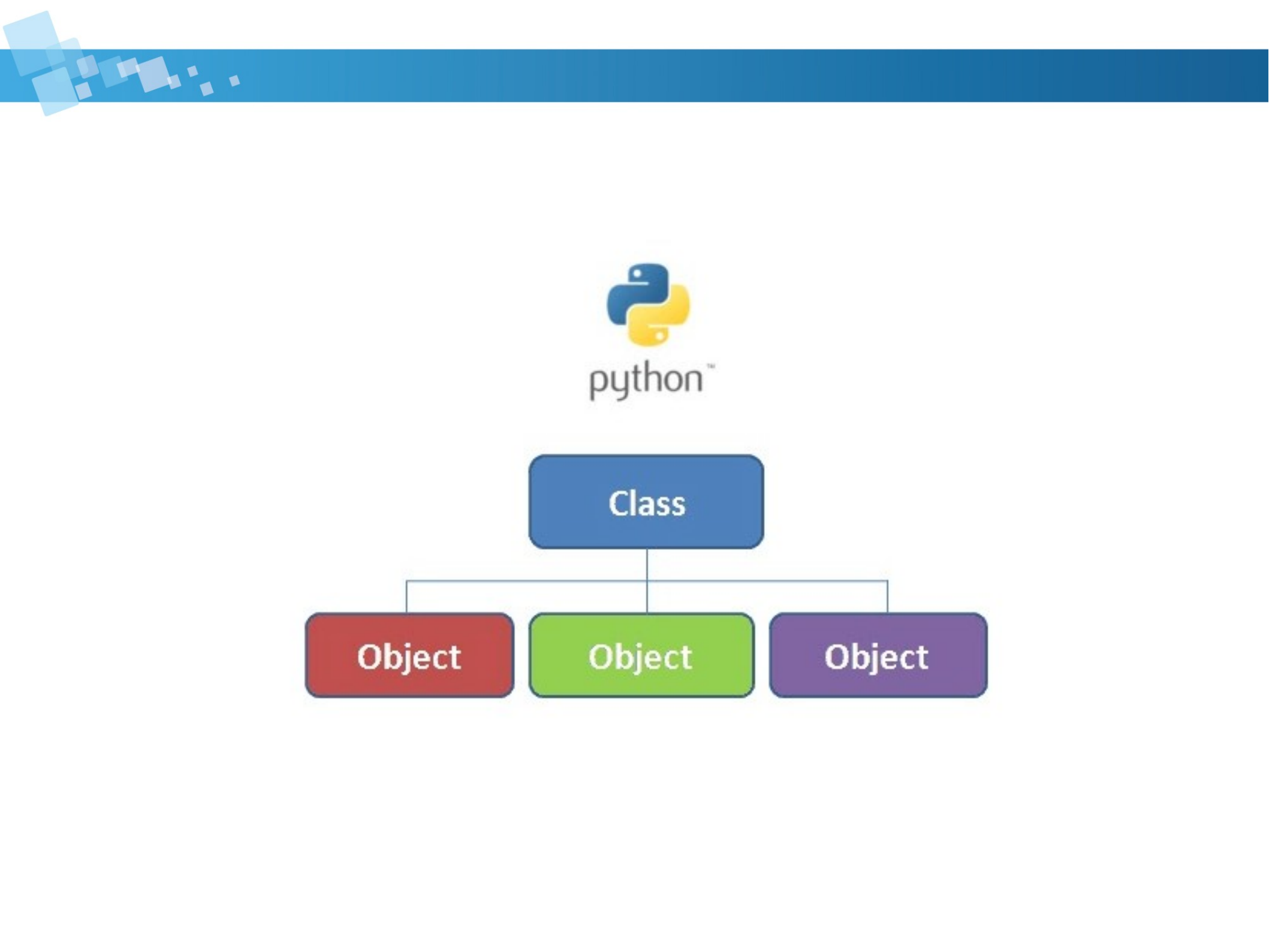
Посылка сообщений (вызов метода)

Объекты взаимодействуют, посылая и получая сообщения. Сообщение — это запрос на выполнение действия, дополненный набором аргументов, которые могут понадобиться при выполнении действия. В ООП посылка сообщения (вызов метода) — это единственный путь передать управление объекту. Если объект должен «отвечать» на это сообщение, то у него должна иметься соответствующий данному сообщению метод. Так же объекты, используя свои методы, могут и сами посылать сообщения другим объектам.


Объект 1
Состояние
Переменная 1
Переменная 2
Интерфейс
Метод 1
Метод 2
Метод 3
Объект 2
Интерфейс
Метод 1
Метод 2
Состояние
Переменная 1
Сообщения

Понятия ООП

- Класс
- Объект
- Интерфейс


python™
Class
Object
Object
Object

Класс

Класс — универсальный, комплексный **тип данных,** состоящий из тематически единого набора «полей» (переменных более элементарных типов) и «методов» (функций для работы с этими полями), то есть он является моделью информационной сущности с внутренним и внешним интерфейсами для оперирования своим содержимым (значениями полей). В частности, в классах широко используются специальные блоки из одного или чаще двух спаренных методов, отвечающих за элементарные операции с определённым полем (интерфейс присваивания и считывания значения), которые имитируют непосредственный доступ к полю.

Объект (экземпляр)

Объект - это отдельный представитель класса, имеющий конкретное состояние и поведение, полностью определяемое классом. Каждый объект имеет конкретные значения атрибутов и методы, работающие с этими значениями на основе правил, заданных в классе.

Интерфейс

Интерфейс - это набор методов класса, доступных для использования. Интерфейсом класса будет являться набор всех его публичных методов в совокупности с набором публичных атрибутов. По сути, интерфейс специфицирует класс, чётко определяя все возможные действия над ним.

Класс, объект

Когда речь идёт об объектно-ориентированном программирование в Python, первое, что нужно сказать – это то что любые переменные любых типов данных являются объектами, а типы являются классами.

Каждый объект имеет набор атрибутов, к которым можно получить доступ с помощью оператора “.”. Мы уже имеем опыта работы с атрибутами, пример списков:

```
lst = [1, 2, 3]

lst.append(4)
```

Атрибуты, которые являются функциями, называются методами.

Создание класса CLASS

Создать класс можно с помощью конструкции **class**

```
# Создание пустого класса Animal
class Animal:
    pass
```

Создание объекта класса

Для того чтобы создать объект на основе класса Animal нужно вызвать его как функцию.

```
animal = Animal()
```

Определение отношения объкта

Для определения, к какому классу относится объект, можно вызвать внутренний атрибут объекта `__class__`. Также можно воспользоваться функцией `isinstance` (рекомендуется этот вариант).

```
# Определить класс объекта
print(isinstance(animal, Animal))

print(animal.__class__)


# Поля объекта

print(dir(animal))
```

Класс с атрибутами

```
class Animal:
    color = "red"
    weight = 200

animal = Animal()


# Поля объекта
print(dir(animal))

print(animal.color)

print(animal.weight)
```

Изменение атрибутов

```
class Animal:
    color = "red"
    weight = 200

a1 = Animal()
a2 = Animal()

a1.weight = 400
print(a1.color, a1.weight)
a2.color = "blue"
print(a2.color, a2.weight)
```

Создание новых атрибутов внутри объектов

```
# Добавим новый атрибут speed к объектам a1, a2
class Animal:
    color = "red"
    weight = 200

a1 = Animal()
a2 = Animal()

a1.speed = 300
a2.speed = 400


print(a1.color, a1.weight, a1.speed)
print(a2.color, a2.weight, a2.speed)
```

Доступ к атрибутам класса

Класс Animal так же является объектом и внутри него так же есть атрибуты, как и внутри объектов, созданных с помощью класса Animal.

```
print(Animal.color, Animal.weight)

Animal.color = "green"

Animal.weight = 300

print(a1.color, a1.weight, a1.speed)

print(a2.color, a2.weight, a2.speed)
```

Как вы могли заметить, в объектах изменились значения только тех атрибутов, которые мы не меняли до этого в этих объектах. Это поведение будет объяснено позже.

Методы класса

Для объявления методов внутри класса нужно просто создать функцию внутри класса. Эта функция должна принимать как минимум один аргумент. По общепринятому соглашению этот аргумент называется **self** и в него передаётся сам объект класса, из которого этот метод и вызывается.

```
class Animal:
    def set(self, value):
        self.value = value

    def print(self):
        print("VALUE:", self.value)

animal = Animal()
animal.set(20)
animal.print()
animal.set("May..")
animal.print()
```

Конструктор __init__

В классах можно описывать так называемые "волшебные" методы, то есть методы, которые имеют заранее прописанный функционал в языке Python. Одним из таких методов является конструктор.

Конструктор – специальный метод, который вызывается сразу же после создания объекта и производит первичные действия. Чтобы создать конструктор в Python – нужно создать функции в классе с именем __init__.

```
class Animal:
    def __init__(self, color, weight):
        self.color = color
        self.weight = weight

a1 = Animal("red", 300)    # при создании объекта нужно
a2 = Animal("green", 200)  # передавать все аргументы, кроме
                           #self, которые принимаются в
                             конструкторе.
a1.color = "blue"
a2.weight = 500
print(a1.color, a1.weight)
print(a2.color, a2.weight)
```

Деструктор __del__

Также существует такой "волшебный" метод который называется деструктор. Он вызывается в момент удаления объекта из памяти.

```
class Animal:

    def __init__(self, name):

        self.name = name


    def __del__(self):
        print(self.name, "will deleted.")

a1 = Animal("Kent")
a2 = Animal("Afon")
a3 = Animal("Zuk")
del a2                  # Удаление переменной
```

Сначала на экран вывелась строка об удалении Afon, а потом об удалении остальных объектов. Несмотря на то, что мы явно их не удаляем, они всё равно удаляются при достижении программой конца.

```
class Lection:

    def __init__(self, name):

        self.name = name

    def __del__(self):
        print(self.name, "Продолжение следует.")


obj = lection("lection_17")
del obj
```